RASM

roudoudou Assembler
v0.42

Documentation RASM v0.42 (c) Édouard BERGÉ 2017

« Copyright © BERGÉ Édouard (roudoudou)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation/source files of RASM, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. The Software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the Software. »

Documentation RASM v0.42 (c) Édouard BERGÉ 2017

Documentation RASM v0.42 (c) Édouard BERGÉ 2017

1. Introduction

18 years ago, I programmed an assembler/disassembler called Zasm/dizasm. In fact it was mainly a proof of concept of a mono-pass assembler. I used a little the disassembler with personnal projects. Sources were published but the spread was so confidential that it was hard to retrieve them back on my HDD.

Since then, another assembler called Zasm exists, created by a spectrum developper team.

As the code of Rasm is from scratch (except a few concept), it's the pretext to choose a new name: Rasm.

2. Installation

Rasm a stand-alone executable, there is no installation.

3. Compilation

Linux compilation:

```
cc rasm_v042.c -lm -lrt
mv a.out rasm
strip rasm
```

Windows compilation:

```
cl.exe rasm_v042.c -DOS_WIN -Ox
```

4. Rasm behaviour

Rasm is supposed to be easy to use. It will set default filename, guess what memory region to save or create a cartridge file is ROM bank were selected during assembly.

Rasm allow multiple ORG in the same memory set but it will not allow to write twice the same adress in the same memory set. Each ORG, Rasm check there is no overwrite.

If you need to generate many code portion at the same adress, you have two possibilities. Either you use the <output> parameter of ORG directive (to choose another output adress), or you may create another memory set using directive WRITE DIRECT -1,-1,#C0

When you select ROM with directives BANK or WRITE DIRECT, this is always the same memory set for each ROM. If you need another memory set, do not use ROM.

When using SAVE directive, Rasm disable automatic binary output to file or cartridge. You may want to force CPR writing with directive BUILDCPR.

When possible, Rasm try to display comprehensive error messages to point out the problem.

Rasm is using a pre-processor to normalize code, remove useless spaces, comment, check quotes and legal characters, and finally transform some instructions into others. As an example, XOR,AND,OR,MOD will be converted in a single char like C syntax.

When a read file directive do not refer to an absolute path, le root path is the one from the current file.

5. Using command line

-i <file to assemble> this is the only mandatory option.

-s export symbols to Rasm format
-sp export symbols to Pasmo format
-sw export symbols to Winape format

-sl additionnal option to the three previous one. It exports also local symbols

- Arnold emulator can handle any export format
- Winape cannot handle label of hiw own format when they are too long. The debuger will crash.

-l <label file> import symbols before assembling

-v verbose mode

-m maxam calculation style

-o <output prefix> set the base name for default output files (binary, cartridge, symbols). The default value is “rasmoutput”.

6. Code source format

6.1 generalities

Assembleur is not COBOL, it is useless to indent sources with Rasm, except to be pretty. You may use two points to separete labels, or not. Rasm try to be free style. The direct consequence of this free writing style involves a difference with the assemblers of obsolete design. It is not possible (fortunately) to create a label that has the same name as a directive or Z80 instruction.

Files may be read to Unix or Windows format, an internal conversion is done.

Rasm is not case sensible. All chars will be converted upper case. Don't be surprise when Rasm display error messages.

6.2 Comments

Rasm comment starts with semicolon. All chars are ignored until next carriage return.

There is no multiline comment.

6.3 Literal values

Rasm knows the following literals:

- In decimal if the value begin with a digit.
- In binary if the value begin with a %.
- In hexadecimal if the value begin with a #.
- An ascii value if a single character is between quote.
- An internal value (variable or label) if the literal begin with a letter or an '@' for local labels.

Rasm does all its internal calculations in double floating-point precision. A correct rounding is performed at the end of the calculation chain for the integer needs.

Caution, the & character is reserved for the AND operator.

6.4 Autorhized characters

Between quotations, all characters are allowed, at your own risk, for ASCII conversion to Amstrad. outside quotes, you can use all letters, all digits, point, arobas, parentheses, dollar, plus, minus, multiplied, divided, pipe, circumflex, percent, sharp, the paragraph, the rafters and the two types of quotes.

6.5 Files
6.6 Labels

Inside a loop (REPEAT / WHILE) it is possible to use local labels in the same way as with the integrated Winape assembler prefixing the label with the character '@'.

For each loop iteration, for each loop nesting, a suffix is added to the local label containing the hexadecimal value of the internal repetition counter. It is thus possible to call a local label to a repetition outside the loop, but this use is not advised.

It's possible to declare a label starting by a dot. This is an old declaration style. The call to this label must be done without the starting dot.

7. Expressions

Rasm use a simple expression engine how can handle the following operators/functions:

- * multiplication
- / division
- + addition
- - substraction
- & boolean operator AND
- | boolean operator OR
- ^ boolean operator OR exclusive
- § modulo
- AND, OR, XOR, MOD inline maxam style
- sin() sinus
- cos() cosinus
- asin() arc-sinus
- acos() arc-cosinus
- atan() arc-tangente
- int() double to int conversion
- abs() absolute value
- ln() natural logarithm
- log10() logarithm base 10
- exp() exponential
- sqrt() square root

Comparison operators:

- == equality (only one = in maxam compatibility mode)
- != different from
- <= less than or equal to
- >= greater than or equal to
- < less than
- > greater than

8. Static variables or alias

It's possible to create aliases with EQU directive. Thoses aliases can't be modified.

9. Dynamic variables

Rasm may use an unlimited number of variables for internal calculations.

Syntax: `mavariable=5` or `LET mavariable=5`

Thoses variables may be used as loop counter, offset or value in a loop or the main code.

Example:

```
dep=0
repeat 16
ld (ix+dep),a
dep=dep+8
rend

ang=0
repeat 256
defb 127*sin(ang)
ang=ang+360/256
rend
```

10. Directives

9.1 EQU

Create an alias

Example:
```
myvar EQU 5
myothervar EQU myvar*2
```

9.2 ALIGN <boundary>

Align code to a given boundary

Examples:
`ALIGN 2` to align to even adress
`ALIGN 256` to align to the high byte adress

9.3 AMSDOS

Add an Amsdos header to the automatic binary file. Note: There is no Amsdos header added with SAVE directive.

9.4 BUILDCPR

Force cartridge output when using SAVE directive.

9.5 BANK <rom number>

Select a ROM for assembling. Even if a ROM size is 16K, you can start to write anywhere in the ROM. Rasm will use the first outputed byte as start adress when creating CPR file. ROM number range is from 0 to 31.

Note: When selecting twice the same ROM, a new memory space is created. If you want to create a CPR file, only the first BANK <n> will be used for each ROM.

9.6 IF, IFNOT, ELSE, ENDIF

Conditionnal assembly

Example:

```
CODE_PRODUCTION=1
[...]
IF CODE_PRODUCTION
OR #80
ELSE
PRINT 'Test version'
ENDIF
```

9.7 LZ48 / LZ49

Open a LZ48/LZ49 crunched code part. The output code will be crunched after assembly and following code will be relocated.

It's not possible to call a label located after the crunched code, from the crunched code as we cannot know how big will be the crunched code. There will be an error if you try so.

Limitations:

- It's not possible to assemble a code bigger than 64ko before crunch.
- It's not possible to embed portions of crunched code.

9.8 LZCLOSE

Close a crunched part opened with LZ48 or LZ49

9.9 READ / INCLUDE 'file to read'

Read a text file and include it into the current source code. Relative path starts from the absolute location of the current file. An absolute path do not care about this.

There is not recursion limit. Be aware of what you do.

9.10 INCBIN 'file to read'[,offset[,size[,extended offset]]]

Include a binary file. Read data will be directly outputed. Parameters are like Winape syntax except first offset is not limited to 64K. Extended offset is here for compatibility only.

9.11 INCL48 / INCL49 'file to read'

Read a binary file, crunch it with LZ48 or LZ49 and directly output in the memory.

9.12 LIMIT <maximum adress>

Set a lower limit (other than 64K) for outputed code. If you want to protect a zone, you may use PROTECT directive.

9.13 ORG <code adress>[,<output adress>]

Assemble at a given adress. You may set an output adress to generate the code at a different memory adress.

9.14 PROTECT <start adress>,<end adress>

Protect memory region in the current memory set. Any write to this region will trigger an error and a message.

9.15 STOP

Stop assembling.

9.16 PRINT

Write text/variables during assembling.

9.17 NOLIST / LET / RUN <adress> / BRK

Directives are ignored, only for compatibility with Winape.

9.18 WHILE / WEND

Repeat a block of instructions everytime the condition is true.

Example:

```
cpt=10
while cpt>0
ldi
cpt=cpt-1
wend
```

9.19 REPEAT <n> / UNTIL | REPEAT / UNTIL <condition>

Repeat a block of instructions. You may set a fixed number of repetition or set a conditionnal repeat.

Examples:

```
repeat 16
ldi
rend

cpt=10
repeat
ldi
cpt=cpt-1
until cpt>0
```

9.20 WRITE DIRECT <lower rom>,<upper rom>,<RAM>

If you set a ROM number (lower from 0 to 7, upper from 0 to 31), the directive will have the very same effect as BANK directive.

If you set the RAM (disabling ROM with -1 both lower and upper ROM), Rasm create a new memory set for assembling. Using multiple WRITE DIRECT you can assemble many code at the same adress, but different memory sets.

Example:

```
;default memory space, ORG at zero
defs 65536,0
WRITE DIRECT -1,-1,#C0
defs 65536,0
; new memory space, no matter the RAM bank
; in this example twice #C0
WRITE DIRECT -1,-1,#C0
; no error since it's a third memory space
defs 65536,0
```

Rasm created three memory spaces. The default memory space and two additionnal set for each WRITE DIRECT call. There is no limit for memory space, except your system memory.

When a new memory space is creaeted, you cannot get back to any previous one.

9.21 SAVE 'binary file to write',<adress>,<size>

Write a binary file starting from adress of the current memory set to adress+size. Note that the saves are done only if there is no error at the end of assembly.

9.22 CHARSET 'string',<value> | <byte>,<value> | <start>,<end>,<value>

Allows the value of characters or characters in strings to be assigned alternative values. There are 4 forms of this directive.

- 'string',<value> First character of the string will have a new ascii <value>. Next char will have <value>+1 etc. until the end of the string.
- <byte>,<value> set a new <value> for the character <byte>
- <start>,<end>,<value> Set from the character <start> until <end> an incremental value starting from <value>.
- “no parameter” will set default values to all characters.

This directive is compatible with Winape.

9.23 MACRO / MEND

See macro chapter.

11. Macros

Rasm handle macros with curly brackets (Winape compatible). It is possible to do conditionnal assembling inside macros because each macro call insert a brand new code with new substitutions of parameters. Then the code is interpreted like if there were no macro.

Example for a long distance LD (IX+offset),register write:

```
macro LDIXREG register,dep
if {dep}<-128
	push bc
	ld bc,{dep}
	add ix,bc
	ld (ix+0),{register}
	pop bc
else
if {dep}>128
	push bc
	ld bc,{dep}
	add ix,bc
	ld (ix+0),{register}
	pop bc
else
	ld (ix+{dep}),{register}
endif
endif
mend
```

12. Instructions

All documented and undocumented instructions are supported.

Index registers IX and IY 8-bits form may be used with ixl, lx ou xl, etc.

Complex instructions syntax:

```
res 0,(ix+0),a
bit 0,(ix+0),a
sll 0,(ix+0),a
rl  0,(ix+0),a
rr  0,(ix+0),a
```

Input/Output undocumented instructions syntax:

```
out (<n>),a ; <n> is a 8 bits value
in a,(<n>)
in 0,(c) or in f,(c)
```

13. Limitations

- It's not possible to use instruction or directive as label.
- Crunched section cannot exceed 64K before crunch.